International Journal of Multidisciplinary Research and Growth Evaluation.

The CFAR Radar Target Detector HW/SW Architecture Implemented on an FPGA

Riyadh Abdulhamza Al-Alwani
Department of Electrical Engineering, College of Engineering, University of Babylon, Babylon, Iraq

* Corresponding Author: **Riyadh Abdulhamza Al-Alwani**
* Corresponding Author's Email: alwaniriyad@gmail.com

Article Info

ISSN (online): 2582-7138
Volume: 04
Issue: 03
May-June 2023
Received: 02-04-2023;
Accepted: 19-04-2023
Page No: 243-248

Abstract
This paper describes the HW/SW Codesign FPGA-based architecture of a log-normally distributed B-ACOSD CFAR target detector for radar systems in the abstract. The complete CFAR system is evaluated to see which components need to be strengthened so that the detection procedure can be carried out in real time. To ensure the CFAR Architecture is fully optimised, we have devoted careful attention to the Altera environment's custom instruction strategy. The CFAR detector is a hybrid hardware/software setup. The Avalon switch fabric allows the hardware modules to talk to the NIOS II processor, which runs the software. The on-chip memory and custom logic components complement the universal asynchronous receiver/transmitter (UART) and JTAG (JTAG) interfaces. The suggested SoC is verified and tested with the help of an Altera Stratix IV EP4SGX230KF4C2 running at 250MHz.
We improved the efficiency of our embedded target detection system by fusing hardware and software approaches. When compared to a software-only solution, this resulted in a 0.45µs reduction in overall latency.

Keywords: FPGA, HW/SW, CFAR, Architecture

1. Introduction

The received signal is processed by a radar system, which then determines the target type (target or clutter) and the positions of the detected items. After then, the data is shown on the screen. Whenever the signal strength is greater than a predetermined threshold, regardless of whether the background is cluttered or absent, the target can be easily localised. In real-world situations, the echo is often accompanied with time- and location-varying background noise. Target extraction is more effective when the threshold is dynamically rather than statically determined, taking into account local noise and clutter. Adaptive signal processing utilising a configurable detection threshold is necessary to determine if the tested cell is a target [1].

In order to maintain a constant false alarm rate (CFAR) in radar systems, several approaches have been presented in the literature [2, 3], including cell averaging (CA) and ordered statistics (OS). As mentioned in [4], one such device is the OS-CFAR detector. This detector takes the noise level around it as a reference and adjusts accordingly. In conditions with homogenous backgrounds, the OS-CFAR detector can differentiate closely spaced interferences with just a little increase in detection loss, in contrast to the CA-CFAR detector. Although the CA-CFAR technique takes a lot more time to assess than the CFAR detector, it is the superior CFAR strategy to use in homogeneous settings.

The Censored MeanLevel Detector (CMLD) [5, 6], Greatest-of-CFAR (GO-CFAR) algorithm [5, 7], and Smallest-of-CFAR (SO-CFAR) approach [5, 8] are just a few examples of the many well-developed OS approaches that have been investigated for a variety of purposes. However, as the number of targets changes, the assumption of a constant environment collapses. The CA-CFAR processor's efficiency drops dramatically in these environments.

More robustness against a heterogeneous environment has been achieved by the development of new classes of CFAR approaches [9, 10]. Unfortunately, real-time analysis was not possible because these systems were software-only. However, processing radar data correctly requires keeping the false alarm rate consistent and reasonable.

By comparing the processed target signal to the examined cell using an adaptive threshold detector, a CFAR may be derived. You need to know about probability distributions like the Rayleigh, Weibull, K-, and lognormal distributions to make this comparison. Constraints in terms of real time must also be taken into account, and this includes reducing as much time as possible spent on target identification, particularly for high-resolution targets. Conducting a solution space design inquiry is the only method to guarantee the CFAR detector's success in this mission.

However, CFAR radar detection is still more theoretical than practical in a real-time scenario due to the absence of easily accessible high-computational signal processing methods. System-on-chip (SoC) architecture is a possible option for the real-time CFAR processor due to the time-sensitive nature of target identification by a high-resolution radar system. The CPU, glue logic, and memory are all combined onto a single chip in systems-on-a-chip architecture. SoC manufacturing has been made easier and faster thanks to recent developments in field programmable gate array (FPGA) technology.

In this research, we demonstrate the implementation of the B-ACOSD CFAR algorithm on a Nios II-based FPGA for the detection of censored ordered statistics. This detector should be able to reliably perform automatic target recognition and interference counting inside a lognormal clutter distribution. The CFAR detector runs on an Altera Stratix IV board with a hybrid hardware/software design based on the combination of the NIOS II core processor and bespoke logics written in VHDL language. To provide for efficient data transfer between the various parts of the system-on-chip, the Avalon switch fabric is included into the same FPGA as the rest of the parts.

On the forth page. The suggested CFAR system is well-suited for high-resolution radar applications in arid regions [11], as it serves as a prototypical HW/SW example of how to acquire a processing latency of less than 500 ns.An overview of the remaining sections of this work is provided below. In Section 2, we'll cover the basics of CFAR, while in Section 3, we'll go into studies of how various CFAR algorithms may be implemented in hardware. In Section 3, the B-ACOSD algorithms and formulae are presented. In Section 4, we detail the detectors' SoC HW/SW FPGA architecture. The embedded system's co-simulation findings and implementation for target identification are presented in Section 5. Our last thoughts and suggestions for further study are presented in Section 6.

2. Related work

Some form of detecting mechanism is necessary for a radar system to ascertain whether or not a return originated from a target. When the threshold is too low, more targets are recognised, but there is also an increase in false positives. If the threshold is set too high, however, fewer false alarms will occur at the expense of fewer targets being acknowledged. Most radar detectors have their thresholds set to limit the possibility of false alerts. (Pfa). A false alarm threshold may be determined if the levels of background noise, clutter, and interference are constant throughout time and place. But in reality, the levels of spatial and temporal noise are raised by unwanted clutter and interference sources. To keep the false alarm probability constant, the threshold might be adjusted up or down in real time using an adaptive threshold. The term "CFAR detection" is used to characterise this method.

Figure 1 depicts a fairly typical CFAR processor. A shift register is used to serially arrange the input signals. By applying a CFAR processor to data from neighbouring cells, the adaptive threshold (X0) may be determined. Whether or whether X0 is bigger than the target value is used to make the call. If the value of the tested cell is higher than the threshold value, it is considered to be a target.

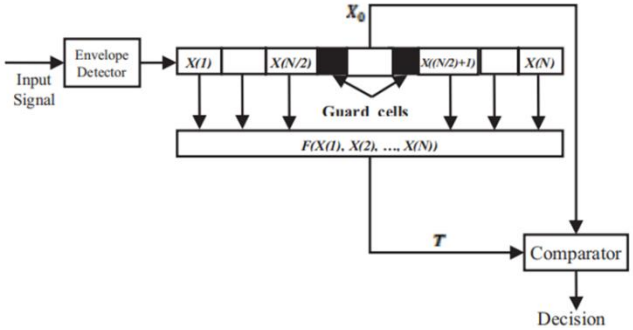

Fig 1: Schematic representation of a common CFAR algorithm

The geometric mean of the reference cells is used by the CA-CFAR detector [3], the simplest and oldest CFAR detector, to establish its adaptive threshold. In recent years, other alternative CFAR algorithms have been created. Based on the clutter power distribution and the interference objectives, three models may be used to categorise CFAR algorithms.

- When the clutter power distribution changes, the likelihood of a false alarm rises, but this may be mitigated by employing the greatest-of-selection logic for the CA-CFAR detector (GO-CFAR) [12]. A hence-CFAR approach using smallest-of-selection logic is recommended for the CA-CFAR detector since the GO-CFAR detector's target detection effectiveness decreases in the presence of several interfering targets. [13].
- The CMLD may be utilised as a target detector when the background clutter consists of uniform white Gaussian noise and distracting targets. The CMLD is able to estimate the noise level by excluding a portion of the noise from the target sample. Furthermore, when the data in the window has been organised, the trimmed mean-level CFAR (TM-CFAR) detector [4] uses trimmed averaging. If you don't know how many false positives to predict, you can use the generalised CMLD (GCMLD) or the OS-CFAR detector (which selects a single ordered sample to reflect the expected noise level in the cell being tested). The generalised two-level CMLD (GTL-CMLD) [14] uses an automatic censoring technique to exclude samples with interfering targets or extended clutter from the reference window of the tested cell. This method is frequently employed whenever there is a shift in the clutter power distribution and interference from several targets.
- Non-Gaussian distributions of clutter are covered in the final presentation. The lognormal, Weibull, gamma, and K-distributions have all been used to describe the non-Gaussian clutter distributions discovered by envelopes. There has been some development in Weibull clutter CFAR detection recently. For example, in [15] we find a presentation and analysis of the performance of the maximum-likelihood CFAR (ML-CFAR) under situations of undetermined size and shape parameters. For this reason, we have developed the optimum Weibull

CFAR (OW-CFAR) approach [16], whereby statistical tests are formulated in terms of an estimate of the mean power of the Weibull clutter.

- Theoretical progress in CFAR detection has outpaced its hardware implementation. Few attempts have been recorded to build CFAR processors in hardware. In particular, the hardware architecture shown in [16-18] is flexible, and may be utilised to deal with noisy signals for target recognition using CFAR algorithms. The architecture was made to be set for Max, Min, and Cell-Average (CA) CFAR algorithms, in addition to allowing parallel/pipeline processing. In [19], we learn about a parallel OS-CFAR implementation. Combining the FPGA architectures for CACFAR and OS-CFAR is what [20] does. The TM-CFAR algorithm is shown to be implemented on an FPGA in [21]. Simple CFAR algorithms, which fared best when the clutter followed a Gaussian distribution, were used in all of these examples.

Automatic Censored Cell Averaging (ACCA) ODV CFAR was developed by Alsuwailem *et al.* [22] as a defence mechanism against this. However, validation is performed in an environment that is not real-time, and dynamic interaction with the underlying architecture is not supported. In addition, the Radar System ecosystem lacks a standard interface for facilitating communication between its many components. The radar sensor of the vehicle utilised by Winkler *et al.* [23] was built on a system-on-chip (SoC) with a programmable central processing unit (CPU). All input/output and user-defined logic is managed by the central processing unit.
Earlier this year, Almarshad *et al.* [24] unveiled an automated censoring detector called the ACOSD CFAR. Automatic filtering of competing targets of unknown number in log-normal noise is a key component of this strategy. As radar resolution has improved, the log-normal distribution has become the preferred model for representing the amplitude of clutter, replacing the less precise Rayleigh distribution. However, when samples are obtained from a log-normal distribution, the automated censoring techniques established for Rayleigh clutter and described in [11, 22] cannot be applied directly. The rest of this study focuses on SoC implementation algorithms. The idea comes from the ACOSD-CFAR [24].

3. The Algorithms for Detecting ACOSD

ACOSD CFAR algorithms include a two-stage detection process: (1) excluding potentially misleading reference cells, and (2) doing the actual detection. Using a properly rated cell set, adaptive thresholds are calculated in real-time, and the unknown background level is estimated. You don't need to know anything about the clutter's make-up or the number of overlapping targets to utilise this detector. These are the parts that make up a CFAR processor: A delay line is tapped to record the radar's outputs. The signal in question is the one in the cell with the subscript. At the conclusion of the CFAR procedure, the tissue is encircled with scaffolding cells. The ACOSD cell-free acidic residue (CFAR) arranges neighbouring cells from smallest to largest by size. We tap into the radar's delay line signal (XI:i=0,1,...,N) and record it. The goal signal is in the test cell, which is denoted by the subscript [I=0]. The remaining neighbouring cells are the CFAR method's necessary auxiliary cells. The ACOSD CFAR ranks the N neighbouring cells from largest to smallest.

$$X(1) \leq X(2) \leq \text{------} \leq X(N) \quad (1)$$

Following cell sorting, cells go to the detection step. There are currently two distinct algorithms used by B-ACOSD and F-ACOSD. In the B-ACOSD method, the sample X(N) is measured against an adaptive threshold (TC0) calculated by a formula.

$$T_{C0}=X(1)^{1-\alpha 0}\, X(p)^{\alpha 0}. \quad (2)$$

The maximum number of observations is denoted by X(P), and the level of false censoring desired by the constant. Detection performance is said to be rather excellent at (p>N/2) according to [26]. In the event that X(N)TC0, then X(N) is a noise-free, clutter-free sample, and the process will stop. If sample X(N) exceeds the threshold TC0, then it is assumed to be the echo back from a misbehaving target. The algorithm now needs to decide if X(N-1) is larger or lower than the threshold value, given that X(N) has been censored.

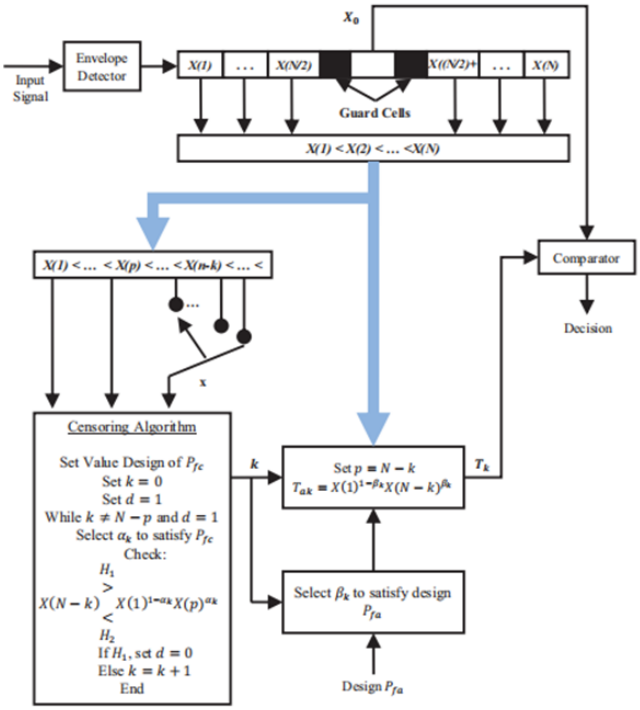

Fig 2: Algorithm block diagram for B-ACOSD

$$T_{ct}=x(1)^{1-\alpha 1}x(p)^{\alpha 1} \quad (3)$$

In order to determine if the given data is an interference-free sample of clutter or a target with interference. The (K+1)th iteration compares the x(N-K) sample against the TCK threshold and makes a decision based on the results.

$$X(N-k) \underset{H_0}{\overset{H_1}{\gtrless}} T_{ck}\,;\, 0 \leq k < N-p \quad \textbf{(4)}$$

Where $T_{ck}=x(1)^{1-\alpha k}x(p)^{\alpha k}$

Clutter samples with interference are shown to be X(N-K+1), X(N-K+2), X(N), and so on, whereas (H0) means that X(N-K) is a clutter sample without interference. The subsequent

tests will proceed on the assumption that H1 is correct. If the cell being probed is found to be homogenous (i.e., clutter sample alone), the procedure terminates; otherwise, it continues until the K=NP highest cells have been evaluated. The block diagram of the B-ACOSD algorithm is shown in Figure 2.

If the tested cell, X0, is larger than the threshold cell, Tak, then detection has occurred.

$$X_0 \underset{H_0}{\overset{H_1}{\gtrless}} T_{ak}\,;\, 0 \le k < N - p + 1 \qquad (5)$$

Both H1 and H0 assume that the target is either present or absent from the test cell, respectively. The cutoff value in B-ACOSD CFAR is

$$T_{ak}=X(1)^{1-Bk}X(N-K)^{Bk} \qquad (6)$$

By choosing K = number of overlapping targets observed during centring phases, the design's Pfa is decreased.

Threshold Values

Choosing an appropriate threshold for the algorithms is essential [26]. These cutoffs need to be established so as to reduce, as much as possible, the chance of a false positive in a carefully monitored hypothesis test. The thresholds were determined using Monte Carlo simulation with small Pfa and Pfc values.

B-ACOSD's computed threshold parameters k and k for Pfa=0.001 and Pfc=0.01 are shown in Table 1.

Table 1: P_{fa}=0.001 and P_{fc}=0.01 are the threshold values for B-ACOSD.

(N,p)		k					
		1	2	3	4	5	6
(16,12)	α_k	2.596	2.038	1.709	1.443	-	-
	β_k	1.635	1.889	2.12	2.37	2.64	-

4. B-ACOSD CFAR architecture HW/SW development using FPGAS

A-*Hardware and Software Framework for Radar Systems*

A Nios II fast processor was integrated into an FPGA board (Altera Stratix IV) to carry out the B-ACOSD algorithm.

Hardware modules developed with VHDL code and the custom instruction technique, and software modules given in ANSI C. To speed up time-critical software calculations, we actually add to the Nios II processor's instruction set with specific instructions. Because the ALU of the Nios II processor is located near a proprietary logic component, developers may combine many standard instructions into a single one for hardware execution, saving both time and space. The computationally expensive software of the CFAR programme is optimised with the use of custom instruction, and not only the outer loops. We may modify the Nios II core processor on our system-on-chip with the help of supplementary instructions so that it meets the stringent timing requirements of the B-ACOSD design.

The components of the embedded CFAR architecture are shown in Figure.3.

Nios II core processor with Master port, 8 KB data cache, 16 KB instruction cache, and 32 bits of data route and software speed.

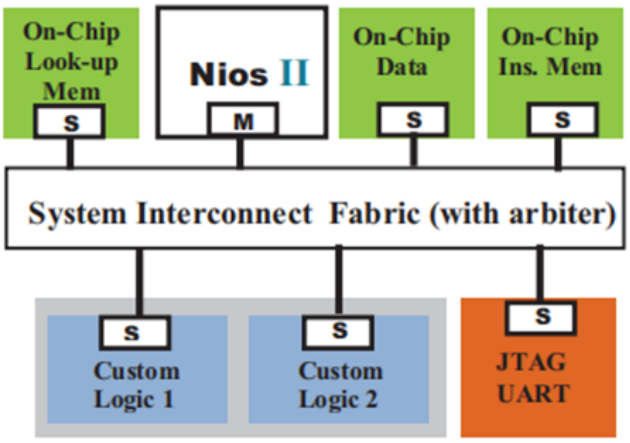

Fig 3: Embedded System with B-ACOSD based on Nios II

- Our embedded system relies on the Avalon bus, which is incompatible with any hardware components not defined in HDL language at the RTL level and sporting only slave ports.
- The Avalon Memory Mapped bus is used by the SIF to communicate with the custom instruction, NIOS II processor, and all other interfaces on the same system-on-chip.
- Additional timers and on-chip memory of 128kx32 and 64kx16 provide accurate monitoring of all timing aspects inside the CFAR architecture.
- Connecting to a target, downloading software, and seeing on-chip trace data are all made easier using JTAG UART interfaces.

B. *HW/SW Design Process*

To build the B-ACOSD System on Chip, we propose a conventional HW/SW design strategy, as shown in Figure 4.

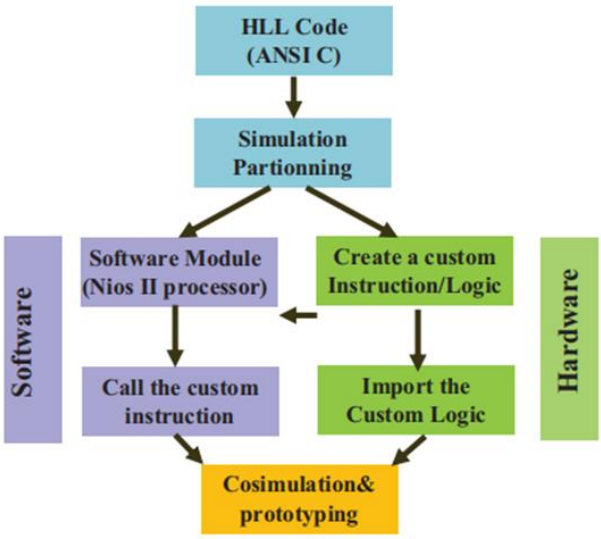

Fig 4: Typical Hardware and Software Design Flow

- The first step is to use an HLL, such as ANSI C, to construct a software architecture from scratch. The FPGA's Nios II processor is executing the software under

microC/OS II to determine whether crucial parts should be exported as hardware.

- Before adding the custom instruction modules into the design, create them in an HDL language like VHDL or Verilog and test them with simulations.
- Add the custom instructions to the SOPC as an internal instruction used by the Nios II processor to include them into the design. You may use either ANSI C or assembly to run these one-of-a-kind instructions.
- • Co-simulate the system using the Nios II, bespoke logics, embedded memory, and other interfaces available on the stratix IV prototype Board.

5. Field Testing and Verification

Depending on where you draw the line between hardware and software, the B-ACOSD CFAR design, which is based on a single FPGA chip, may or may not look like Figure.3. Our algorithms' less crucial components are executed by the Nios II core processor, while the crucial bits are exported as bespoke logic specified in VHDL. A first-realization pure software core operating on the Nios II processor has been used to calculate execution time for the three primary components, which include sorting, filtering, and identifying modules. Second, we used the Avalon interface to link the Nios II CPU to the components listed in the VHDL file. Below is a breakdown of how long it takes to run the B-ACOSD design on a Stratix IV board.

Table 2: B-ACOSD software module execution time

B-ACOSD Modules	Delays in µs
Sorting module	18
Censoring module	104
Detection module	21
Total delay	143

According to our findings, the most critical part of the procedure is identifying the cutoff computation for the filtering module and a subset of the detection module. The sorting mechanism also plays an important role. In order to incorporate the filtering and sorting modules into the detection, we opt to send two user-defined commands. The rest of the code has been updated to work with Nios II core processors. To improve the B-ACOSD's sorting algorithm and lookup table configuration without raising the constant false alarm rate, we tried decreasing the accuracy and the quantity of on-chip memory used during the log function computation.

Hardware implementations of the exponential equations (5, 7) are challenging and expensive due to the necessity for floating point computing. Equations (5) and (7) are logarithmized as follows to reduce hardware requirements and calculation time.

$$\log T_{ck} = (1-\alpha_k)\cdot \log X(1) + \alpha_k \cdot \log X(p) \quad (7)$$

$$\log T_{ak} = (1-\beta_k)\cdot \log X(1) + \beta_k \cdot \log X(N-k) \quad (8)$$

In this case, the multiplication step in computing power is replaced by an addition. Since hardware implementations of logarithmic computing are complex and time-consuming, this overhead can be reduced by resorting to a lookup table. The lookup table is based on actual observations of real-world radar input data, and it represents lognormal distributions with parameters =1 and =1.1, as stated in [26]. Following the current trend towards logarithmic notation, we transformed the values of our test cells using a LUT that can handle up to 2000 numbers. This logarithmic transformation is likewise accomplished using the aforementioned conversion table. The lookup table is stored in the FPGA's 32K ROM, which is located directly on the device. The chip's 32 kilobytes of ROM can store data with a resolution of 0.0610 bits. This MATLAB simulation of BACOSD CFAR censoring at a fixed resolution yields findings that agree with the real-number version.

Due to its lowered delay in software implementation (as shown in table 3) and its hardware implementation (which provides a high degree of parallelism and an interesting timing), we have changed a Parallel Range Computing (PRC) strategy for bubble sorting to speed up execution time.

Table 3: Execution time of several sorting strategies on the Nios II processor

Sorting technique	Bubble	Even/Odd	PRC
Nios II Execution time	18 µs	10 µs	1.28 µs

We have integrated the hardware and software components of the design, performed simulations, and studied the complexity differences between the two variations with and without the unique instruction. The A-COSD detector's execution complexity employing only the Nios II core processor from the FPAG download is shown in Table 4. Table 5 provided a summary of the system's hardware, including information on the combinational LUTs, logic registers, and On-chip memory. The complexity of the design grows noticeably when the custom logics are implemented as discrete hardware components. As processing time decreases from 143 µs to 0.45 µs, people will spend less time waiting.

Table 4: Powerful Nios II CPUs (100 percent Software Answer)

	Without Custom Inst.	With Custom Inst.
Comb.LUTs	1474 (0.8%)	1683 (1%)
Logic Reg.	1254(0.7%)	1257 (0.7%)
On-chip Memory	94528 (7%)	94528 (7%)

Table 5: SoC hardware and software for B-ACOSD

	Without Custom Inst.	With Custom Inst.
Comb. LUTs	3368 (2%)	4723 (3%)
Logic Reg.	2486(1%)	3074 (2%)
On-chip Memory	4547584 (31%)	4547400 (31%)

The highest clock speed for the NIOS processor in an N=16, p=12 SoC implemented on an FPGA is 250 MHz. Once the software B-ACOSD was in place, we utilised the Avalon interface to relocate the delay-critical components to a set of custom logic blocs that were then linked to the Nios II CPU. Up to 0.45µs of delay can be removed from the overall layout. Processing speeds of less than half a micro-second are needed for real-time applications [24]. The proposed system-on-chip has a default configuration that includes 16-bit data samples, 16 reference cells, and 2 guard cells for demonstration purposes. 256 samples of data were generated using an exponential distribution to verify the HW/SW design. The

ROM can hold up to 16 x 256 bytes of data. One 256-megabyte RAM is used to store the data.

6. Conclusion

The authors of this study claim to have implemented the B-ACOSD CFAR target detector for lognormal clutter in both hardware and software. The suggested system-on-chip is simple to construct and quick to prototype, which are both advantages. The hardware configuration of the prototype worked well enough in a short enough amount of time to validate the idea of co-design. In an effort to export and develop the hardware components in a timely manner, we investigated the custom instruction approach. A large number of interfaces and components, including NiosII, custom logics, on-chip memory, Avalon switch fabric, and more, are included in the proposed FPGA implementation. The suggested architecture has a latency of 0.45µs for detecting each tested cell, which is significantly less than the 0.5µs needed for real-time detection. The EP4SGX230KF4C2 device in the Stratix IV development kit allowed us to precisely measure and validate all time limitations in the proposed design.

References

1. M Barkat. Signal Detection and Estimation. Norwood, MA: Artech House, 2005.
2. RS Johnson, HM Finn. Adaptive detection mode with threshold control as a function of sampled clutter-level estimates, RCA Review. 1968; 29:414-463.
3. H Rohling. Radar CFAR thresholding in clutter and multiple target situations, IEEE Trans. Aerospace and Electronics Systems. 1983; 19(4):608-621.
4. SA Kassam, PP Gandhi. Analysis of CFAR processors in nonhomogenous background, IEEE Trans. Aerospace and Electronics Systems. 1988; 24(4):427-455.
5. HA Meziani, F Soltani. Performance analysis of some CFAR detectors in homogenous and non-homogenous Pearson-distributed clutter, Signal Processing. 2006; 86:2115-2122.
6. GM Dillard, JT Rickard. Adaptive detection algorithms for multiple target situations, IEEE Trans. Aerospace and Electronics Systems. 1977; 13(4):383-343.
7. A Mezache, F Soltani. A novel threshold optimization of ML CFAR detector in Weibull clutter using Fuzzy-neural networks, Signal Processing. 2007; 87:2100-2110.
8. M Barkat, T Larouissi. Performance Analysis of order-statistic CFAR detectors in time diversity systems for partially correlated chi-square targets and multiple target situations, Signal Processing. 2006; 86(7):1617-1631.
9. MA Khalighi, MH Bastani. Adaptive CFAR processor for nonhomogenous environemnt, IEEE Trans. Aerospace and Electronics Systems. 2000; 36(3):889-897.
10. P Henttu, M Juntti, H Saarnisaari. Iterative multidimensional impulse detectors for communications based on the classical diagnostic methods, IEEE Trans. Communication. 2005; 53(3):395-398.
11. S Alshebeili, SM Alhumaidi, AM Obied, YM Seddiq. FPGA Based Implementation of a CFAR Processor using Batcher's sort and LUT arithmetic, in 4th International Design and Test Workshop (IDT),Riyadh-KSA, 2009, 1-6.
12. JH Sawyers, VG Hansen. Detectability loss due to greatest of selection in a cell-averaging CFAR, IEEE Trans. Aerospace and Electronics Systems. 1980; 16:115-118.
13. M Weiss. Analysis od some modified cell-averaging CFAR processors in multiple target situations," IEEE Trans. Aerospace and Electronics Systems. 1982; 15(1):102-114.
14. SD Himonas, PK Varshney, M Barkat. CFAR detection for multiple target situations, IEE Proceeding, Part F: Radar and Signal Processin. 1989; 136(5):193-210.
15. R Ravid, N Levanon. Maximum-likelihood CFAR for Weibull background, [16] R. Ravid and N. Levanon, “Maximum-likelihood CFAR for Weibull background,” IEE Proceeding, Part F: Radar and Signal Processing. 1992; 139(3):256-264.
16. V Anastassopoulos, G Lampropoulos. Optimal CFAR detection in Weibull clutter, [17] V. Anastassopoulos and G. Lampropoulos, Optima IEEE Trans. Aerospace and Electronic System. 1995; 31(1):52-64.
17. C Torres, S Lopez, R Cumplido. A configurable FPGA-based Hardware Architecture for Adaptive Processing of Noisy Signals for Target Detection Based on Constant False Alarm Rate (CFAR) Algorithms, in Global Signal Processing Conference, Santa Clara CA, 2004, 214-218.
18. ML Bencheikh, B Magaz. An Efficient FPGA Implementation of the OS-CFAR Processor, in International Radar Symposium, Wroclaw, 2008, 1-4.
19. R Cumplido, C Uribe, F Del, Campo R Perez. A versatile hardware architecture for a constant false alarm rat processor based on a linear insertion sorter, Digital Signal Processing. 2010; 20:1733-1747.
20. JK Ali, ZT Yassen, TR Saed. An FPGA-based implementation of CA-CFAR processor, Asian Journal of Information Technology. 2007; 6(4):511-514.
21. AM Alsuwailem, SA Alshebeili, M Alamar. Design and implementation of a configurable real-time FPGA-based TM-CFAR processor for radar target detection, Journal of Active and Passive Electronic Devices. 2008; 3(3-4):241-256.
22. AM Alsuwailem, MH Alhowaish, SA Alshebeili, SM Qasim. Field programmable gate array-based design and realization of automatic censored cell averaging constant false alarm rate detector based on ordered data variability, IET Circuits, Devices & Systems. 2009; 3(1):12-21.
23. J Detlefsen, U Siart, J Buchlert, M. Wagner, V Winkler. FPGA based signal processing of an automotive radar sensor, in First European Radar Conference, Amsterdam, 2004, 245-248.
24. M Barkat, SA Alshebeili, MN Almarshad. A Monte Carlo simulation for two novel automatic censoring techniques of radar interfering targets in log-normal clutter, Signal Processing. 2007; 88(3):719-732.
25. R Djemal. A real-time FPGA-based implementation of target detection technique in non-homogenous environement, in Design and Technology of Integrated System in Nanoscale Era (DTIS), Hammamet- Tunisia, 2010, 1-6.
26. R Djemal, S Alshebeili, I Rosyadi. Design and Implementation of Real-time Automatic Censoring Systen on Chip for Radar Detection, in World Academic of Science, Engineering and Technology (WASET), Penang - Malaysia, 2009, 318-324.